A Data Warehouse Using the SAS® System
Harriet Janes, SAS Institute Inc., Cary, NC
Shirley Dixon, SAS Institute Inc., Cary, NC
Terry Lewis, SAS Institute Inc., Cary, NC

ABSTRACT

The SAS System is ideally suited for building a data warehouse. This paper describes the experiences of two Information Center application developers in the development of a data warehouse for use in Decision Support at SAS Institute. Specific techniques and SAS software products used in building the data warehouse and the challenges met during development are discussed. Also included in this paper is a discussion of a new product, the SAS/Warehouse Administrator**, that provides additional capabilities not currently available in other products of the SAS System.

INTRODUCTION

SAS Institute prides itself on using its own technology to run the Institute’s business. The Management Information Systems (MIS) Department at SAS Institute uses the SAS System exclusively, to produce the applications necessary for the business operations of our company. MIS also supports several hundred SAS applications used in many areas of SAS Institute including: Professional Services, Sales and Marketing, Distribution, Book Sales, Publications, Human Resources, Facilities, Video Production, Corporate Communications, Contracts, and Finance. While sometimes features of the operating system software for the platform under which the application runs are taken advantage of, the actual tools used to develop and run the applications are the wide variety of SAS software products that are available to the customers of SAS Institute.

The Management Information Systems Department in the Information Systems Division has a staff made up of four applications programming groups and one information center group. The development and support of the applications used for business operations falls to the four applications programming groups. Although not part of the organizational structure of the MIS Department in Cary, other MIS groups in SAS Institute’s regional and international offices sometimes share applications developed by each other. With more than 2,200 worldwide users at SAS Institute and hundreds of distinct applications, MIS fully exercises most of the SAS software products that make up the SAS System.

BACKGROUND OF THE DATA WAREHOUSE PROJECT

The manager of the MIS Department was given the task of developing a data warehouse for SAS Institute. SAS Institute’s top management had asked that the MIS Department build this data warehouse in a very timely manner. The department manager gave this project to one of her group managers at the end of May, 1995. This manager and an applications developer within her group completed the project by the first week of August, 1995.

What is a Data Warehouse?

At the time, we found only one article about data warehousing. An article from Prism Solutions, "What is a Data Warehouse?", centered around the premise that a "Data Warehouse is a: subject-oriented, integrated, time-variant, nonvolatile collection of data in support of management's decision making process." As much information as possible was gathered from that one article, and a strategy began to be planned for developing the data warehouse.

How to Begin?

A meeting was held with the Institute’s top management to discuss what should be included in the data warehouse for SAS Institute. The CEO said that he wanted to be able to browse the data and also be able to get information about the data and its variables (similar to what is presented in the output from the CONTENTS procedure). He also pointed out programs that he had already written and additional reports that he wanted included in the data warehouse. One thing that was emphasized repeatedly in the meeting...the data warehouse application should be simple to use.

After this meeting, the article from Prism Solutions was reread and reflections were made on what the article described to be the essential components of a data warehouse: metadata, current detail data, older detail data, lightly summarized data, and highly summarized data. These were the same components that management had requested to be included in the data warehouse to be developed for the Institute.

One feature of a data warehouse that management said was not needed for our data warehouse was an extensive EIS (executive information system) type of facility so that users could actually do their own querying of the data. The data warehouse at SAS Institute was simply to be a central place to go when you need to run your own reports or develop your own EIS applications.

HISTORY OF THE DEVELOPMENT OF DATA INFRASTRUCTURE FOR WAREHOUSE

How to Organize the Data?

Now that the basic components of a data warehouse were understood, a plan was made on how the data should be organized in the data warehouse. At this stage of development, decisions were made on how to organize and maintain the data based on current reports being generated from the operational data. Decisions had to be made about what the best platform would be to store all the programs needed to maintain the data. Since most of the operational data resided on the mainframe and also because there were features about the MVS operating system that would help simplify the administration of the data warehouse, the MVS platform was chosen. Even though the programs to maintain the data would reside on MVS, by using SAS/CONNECT™ software it would be an easy task to move the data to whichever platform was chosen as the location for the data warehouse application itself.

Since there were new features of the Orlando Release of the SAS System which are not available on the mainframe, it was decided that the UNIX platform would be where the visual part of the data warehouse application would reside. Data would be accessed through a SAS/SHARE® software server so that all the data in the data warehouse could be accessed across all platforms throughout SAS Institute.

What Data to Include?

After decisions were made on data organization, data subjects were selected that would appear in the data warehouse. It was decided that subjects would be initially based on the sources of data for all reports being generated for top management. The subjects of these reports basically consisted of the customer database and the invoice register. Later other sources were added, but the data warehouse began with just these two data subjects. Because of this type of subject orientation, each of these data subjects would be stored as separate SAS data libraries. Unlike some other data warehouses, there was no need at this point to try to combine all the various data subjects into one metadata base. This would have required the need for data cleansing and reconciliation tools which were still being developed by SAS Institute’s data warehouse product development staff.
Try to keep data maintenance simple—standards developed

Since one of the main goals for the data warehouse was simplicity, there needed to be an orderly naming-convention standard developed for data sets as well as for all the SAS programs associated with them. By using the same naming conventions for all members of the partitioned data sets where the programs were stored and the actual SAS data set names, job maintenance would be a relatively easy task for the Data Warehouse Administrator (DWA). The same naming conventions were followed for all the various macros associated with the programs used in data administration. The naming convention chosen used the first three characters of the name as an abbreviation for the data library to which the data belongs and the remaining five characters of the name to define a unique data set, macro, or partitioned data set member.

Using a naming-convention standard is really important for the Data Warehouse Administrator's job. By knowing the naming convention, it is a very easy task for the DWA to track down problems or answer questions about the data in the data warehouse. If a call comes to the DWA about a possible problem or even a question about a data set in the data warehouse, the DWA can ask the caller for the name of the data set in question. The caller can give that information by clicking on the INFORMATION button located in the application. Just by knowing the prefix of the data set name, the DWA can locate the job used to create it. By using the entire name, the source used to create the data can be located.

To also aid in data administration, a library of SAS macros was built. Macros are called within each program to build and transport the data. Each of the jobs used to maintain all the data sets for each data source consists mostly of a few macro calls from this library. Using macros makes the flow of the jobs easy to understand and follow.

DATA-DRIVEN ASPECTS OF THE DATA WAREHOUSE

Overview

Since one of the goals of the data warehouse project was to be data driven, there needed to be a way to display the data for users without constant intervention from the visual display developer whenever changes were made to the data contained within the data warehouse. To accomplish this goal, the DWA monitors and updates two SAS data sets for each data subject along with some external (non-SAS) files used to create the metadata.

These two SAS data sets contain information necessary to populate the frame where the data is displayed. Each of the data subjects have both of these data sets associated with them. These two SAS data sets are referred to as "indexes". One contains the information about the data sets included in the data warehouse, and the other "index" contains variables in the SAS data sets.

Both of these SAS data sets are re-created each time the DWA gathers the operational data and downloads it to the UNIX platform where the data for the data warehouse resides. One interesting note is that if some quick update needs to be performed on any of the data in the data warehouse during normal business hours, SAS/CONNECT and SAS/SHARE software allows this task to be accomplished without impacting users of the data warehouse in any way. In addition, if the DWA needs to make changes to one of these two "index" data sets during business hours, a small application, which uses the FSVIEW procedure and resides on the UNIX platform, presents an overview of all the data sets stored under each data subject. The FSVIEW procedure allows the DWA to make changes to these data sets, such as what order the data sets should be displayed in the toolbar or what type of action needs to be taken on the data set being displayed. Once changes are made in the "index" data sets, the data is merely downloaded to the UNIX platform with its new updates, and the user simply exits the data set being accessed back to the menu screen and then reaccesses the data to see the changes made by the DWA if the user has been viewing data where the change was made.

At run time, the visual display developer retrieves the values stored in these data sets and uses them to display the data either as the data appears in the data set or in a summarized fashion.

Index of SAS Data Sets

One of the two index SAS data sets contains information about each SAS data set stored under each data subject. This data set, built from the data stored in an MVS partitioned data set member, contains the information needed to run the frame that displays the data and is used to dynamically create the toolbar where data sets within each data subject are selected for viewing. A member exists in the MVS external partitioned data file for each data set stored in the data warehouse and each member name is the same as the SAS data set name.

The variables of interest in this data set include:

- MEMBER—the SAS data set name in the data library whose value is used in an SCL variable throughout the SCL program entry used to run the data viewer screen and to populate the toolbar for data set selections.
- DESCRIPT—contains descriptive titles of the data sets that are inserted in the extended entry field across the top of the frame.
- IMAGE—contains the names of the image entries selected by the DWA to be displayed on the individual toolbar buttons.
- LABEL—contains concise descriptions for each of the data sets listed on the individual toolbar buttons. ILabel has been limited to fifteen characters in length.
- ORDER—is the sort order of the data sets on the toolbar.
- ACTION—describes how the data is to be displayed in the Table Editor according to the data model being used and how to summarize the data if it is being displayed in a summarized format.

Index of SAS Data Set Variables

In addition to descriptions of the data warehouse data sets, there also was a need to have information provided about the variables in the data sets being displayed in the Table Editor. The purpose of this data set is to further refine the display of the data and to drive the application that determines how to display the variables in rows and columns in the table.

The variables of interest in this data set include:

- MEMBER—is the SAS data set name that was selected from the toolbar. It is used for subsetting purposes when a data set is selected for viewing.
- VARNAME—contains the names of each variable in each data set.
- ACTION—describes what is going to happen to the variable in question. For example, some variables may be designated as ones to "hide" from the display; other variables may be designated as ones to "hold" in the display so that they are always visible when scrolling right and left in the Table Editor.
- VARNUM—the variable or position number of the variable in the data set gathered from the CONTENTS procedure. It is used for positioning variables in the Table Editor when displaying data in a table so that columns may be specified by position number so that columns may be held when the table is being scrolled to the right or left.
- VARFMT—contains the variable formats to be used when displaying summarized data in the Table Editor.

The information about the variables for this data set is created whenever the data is gathered from the various operational sources. One nice feature about how this data set is maintained is that any variables that are added and/or dropped from a SAS data set are automatically handled when this data set is created. This is accomplished through one of the features of the CONTENTS
procedure which allows outputting of information into a SAS data set. All types of information are available in the output from this procedure, but the only information needed for this data set was the variable name, format, and its position within the data set.

External file containing metadata

One of the requirements of the application is that there be a way to find out information about the data being selected within the data warehouse. This is referred to as metadata in data warehousing terminology. The metadata that is stored for each data set under any subject is created from an external (non-SAS) file and is displayed using the FSLIST procedure. These external files are stored on MVS as partitioned data set members. There is one member stored for each data set in the data warehouse. The metadata contains basic information such as source of data, support for the operational data, any algorithms used in creating summaries, subsetting criteria, any additional comments that might be helpful in understanding the data, and the SAS data set name along with its corresponding LIBNAME statement.

As the operational data is being processed and moved to the data warehouse, additional information is captured from the CONTENTS procedure and is added to the metadata before being moved to the data warehouse. This is done automatically by the SAS System. The PRINTEO procedure is used to route the output from the CONTENTS procedure to an external file. This file is then appended to the end of the partitioned data set member described previously. The output from the CONTENTS procedure contains information such as when the data set was created, a listing of all variables along with their descriptions, any associated formats and informat and also any indexes associated with the data set.

Since the data in the data warehouse might need to be moved to a different location on a disk volume or even a platform, it was necessary to use a macro call rather than presenting the full path name for the location of the SAS data library.

An example of this appears below.

For example, instead of the LIBNAME statement being:

```
libname nisc '/local/disk0/dw/nisc' server=nnshr02.dw access=readonly;
```

You would use this instead:

```
libname nisc xdata(nisc);
```

Code example for the duda macro:

```
xmacro duda(db);

%unquote('%quote("/local/disk0/dw/db\"\")');

server=nnshr02.dw access=readonly;

%end duda;
```

HISTORY OF THE DEVELOPMENT OF VISUAL DISPLAY

After a plan for the data structure was completed, it was time to plan for the visual display of the data. At this point, another member of the MIS applications development staff was asked to design the visual display for the data warehouse project. A very brief description of what was needed was presented to the developer. It basically consisted of designing a SAS/AF® frame that included descriptions for each data set stored in an extended table. Selections could be made on this frame from this list of descriptions. Once a selection was made, the application would prompt for what type of action to be taken: view the data, display information about the data, or use the QUERY facility on the data selected. One important requirement for the design was that once the design and implementation of the visual display for the data warehouse was completed, it would not be necessary to contact the application developer each time a new data set needed to be added to the data warehouse.

The first prototype was developed in less than one month's time. This prototype was presented to top management. After the presentation, he commented that he wanted to see the data first. He did not want to have to do a lot of clicking on various locations on the frame before the data was displayed. He wanted each data table to be displayed and then click on a button to get more information if he wanted to see more detail on the data. The application developers came away from this meeting thinking that to meet management's wishes, a customized screen would have to be designed for each new data set. This would completely eliminate all the data-driven aspects of the application.

Thanks to the new features of the Orlando Release of the SAS System, the developers were able to come up with a new design that met all the new requirements and still maintained the data-driven features of the first attempt. Also, this redesign was ready within two weeks!

HOW DOES NEW DATA MAKE IT INTO THE DATA WAREHOUSE?

This is the typical way new data makes it into the data warehouse: Once a call comes into the DWA requesting data be added to the data warehouse, the DWA finds out what programmer is responsible for operational data and the programmer gains access to the DWA along with some sample programming code that uses the data requested for the data warehouse. Typically, from the time the DWA gets a phone call and locates the programmer responsible for the data, the new data is ready in one or two hours. One of the best features of storing the data on the UNIX platform is that new data can be added to the data warehouse without affecting users in the application. Again, there is no need to contact the developer responsible for the visual part of the application just because new data has been added.

VISUAL COMPONENTS OF THE DATA WAREHOUSE

SAS/AF software and SAS/EIS® software supplied the functionality needed to deliver to users a graphical-based, easy-to-use application. Using new features available in the Orlando Release of the SAS System, the project goals were accomplished, i.e., providing users with data presented in the desired formats and developing a data-driven application with relatively low programmer intervention and maintenance.

Screen Descriptions

The SAS/AF FRAME entries contained within this application allowed these goals to be met in a timely fashion. Very little screen control language (SCL) was necessary nor was it essential to sub-class any of the FRAME classes provided by the software. Everything needed was available from the beginning.

The following objects were selected to create a GUI-based interface and for their ease of use. They include:

- Image Icon widgets for primary menu selections
- Extended Text Entry widgets for primary label descriptions and push button simulations
- A Toolbar widget for data set selections when viewing the data in the data warehouse
- A Table Editor widget for displaying the data contained within the data warehouse
- Input Field Label widgets for secondary label descriptions
- Input Field widgets for user input

Basically two SAS/AF FRAME entries control this application. The first FRAME entry of importance is the main menu screen where the data warehouse data have been grouped around logical data subjects at SAS Institute. Probably the most important FRAME entry is the one screen that is used to display all of the data in the data warehouse.
The initial Entry Screen

The main menu contains the selections for viewing the data groupings and information about various aspects of the application. Here are examples of image icon widgets, an Extended Text Entry widget used as a descriptive label and several Extended Text Entry widgets (across the bottom of the screen) used to simulate push buttons.

DISPLAY 1. SAS Institute Data Warehouse Main Menu

The Data Viewer Screen

Being entirely data driven, the screen that is used to display the data warehouse data is populated at run time with and by information that is passed to it contained in two SAS data sets, maintained by the DWA. This information tells the FRAME what data to display and how to display it and also provides detailed descriptions of the data. As mentioned earlier, the applications developer responsible for the visual display need not be concerned with these details each time new data is added to the data warehouse. The data-driven aspects will be demonstrated later in this paper.

DISPLAY 2. Data Viewer, Tabular Format without Summarization

DISPLAY 3. Data Viewer, Tabular Format with Summarization

Widgets on the data viewer frame include an Input Label widget, an Input widget, an Extended Text Entry widget for descriptive labeling, several Extended Text Entry widgets for push button simulation, a Toolbar widget for data set selections and the Table Editor widget for displaying the data.

The Table Editor

Crucial to the success of the data warehouse project was the need to display the data in different formats depending upon what the users want. The Table Editor Class allowed this need to be met by giving the developer the control over how the data would be presented to the users.

With the shift towards an object-oriented approach to programming and the model-viewer-controller (MVC) paradigm, the Table Editor provided the flexibility to deliver the data at run time, while freeing the visual display developer from the concern of the underlying data structure. The DWA passes parameters to the visual display developer that are in turn used to display the data accordingly.

In most discussions of MVC, one frequently sees descriptions that refer to "generally having several viewers that may be attached to a single model depending upon the needs of your users". In this application, a reverse approach was taken. The viewer that is being used always remains the same; but different data models are being connected to the viewer at run time when new data sets are selected for observation, depending upon how the data are to be displayed.

From the beginning in the development of the data warehouse, it was known that some data would be displayed as it appeared in the data set and some data would need to be summarized prior to its display in the viewer. At that time, guidance was given in a particular technique to perform summarizations and display them in the Table Editor.

But methodologies can change rapidly. A new way of summarizing data is being developed by SAS Institute: the multidimensional database (MDDB). The MDDB provides the flexibility to display data summaries while including drill-down capabilities. This greatly enhanced the ability to see the data in many different ways and has become a powerful tool in our applications repertoire.

PROGRAMMING THE “DATA-DRIVEN” DATA WAREHOUSE

From the visual perspective of the data warehouse, program control is handled via the two "index" data sets described previously, the ENTRY statement in the SCL program where the data is displayed and SAS macro variables. These are the areas used to change the visual display of the data warehouse. The SCL code that controls the display of the data remains fairly static and maintenance-free unless a user requests a change to the actual visual display.
Selecting a Data Library

When the user selects an area of interest from the menu screen, an SCL label section similar to the one below runs.

```lnx
misc;
      call display(‘indw.dw.viewdata.frame’, ‘misc’);
      return;
```

The parameter in the CALL DISPLAY statement after the display entry will be passed to the VIEWDATA frame in an ENTRY statement that will contain the literal to use when opening data sets within a particular data subject.

Initializing the Data Viewer Frame

The SCL code example contains the completed INIT section of the SCL program. In this section, the index of data set variables is being opened for use throughout the SCL program and demonstrates one of the data-driven aspects of this application, a consistent naming convention. The visual display developer has no knowledge of the data subject being requested for viewing nor any reason to have any knowledge of the data since everything being requested to display the data is passed to the data viewer screen using parameters updated by the DWA.

In addition, the toolbar, used for data set selections within each data subject, is build programmatically via a linked subroutine. This information for the toolbar is stored in an SCL list structure. An example of this list will appear below.

Then, the table is prepared for its initial display, including linking to the appropriate subroutine that will tell the Table Editor how to display the variables in the data set in the columns that will comprise the table.

```lnx
entry dib @ 8;
      init;
      attrlist = makelist();  /* Attributes for table */
      holdlist = makelist();  /* All columns to "HOLD" */
      first = makelist();  /* First column to "HOLD" */
      last = makelist();  /* Last column to "HOLD" */
      
      /*-------- Open the index of data set variables for */
      vdsid = open(trim(dib) || ‘.indexivar’, ‘i’);
      vnum = varnum(vdsid, ‘varname’);
      vnum2 = varnum(vdsid, ‘action’);
      vnum3 = varnum(vdsid, ‘varFmt’);
      vnum4 = varnum(vdsid, ‘varFmt’);
      
      /*-------- Build the Toolbar */
      link setools;
      
      /*-------- Load the data model used to display data */
      tabcl = loadclass(‘SHMHELP_FSP_DATA’);
      rc = setitem(‘attrlist’, ‘y’, ‘CONFORM_COLUMNS’);
      call notify(‘table’, ‘SET_ATTRIBUTES’, ‘attrlist’);
      
      /*-------- Give the description of the data set to */
      /*-------- call notify(‘title’, ‘_SET_TEXT_’, getitem(dlist, 1));
      xact = getitem(dlist, 1);
      
      /*-------- ‘Prime’ the initial display of the table */
      /*-------- Display data as is in the data set or */
      /*-------- summarize it based on ACTION in the INDEX */
      /*-------- of data sets */
      if xact = ‘ ’ then do:
         tabid = instance(tabcl);  /* Create instance of DATA_M */
         call notify(‘table’, ‘ATTACH_TABID’, tabid);
         call sendtabcmd, ‘SET_DATASET’, (vname);
         link check();
      end; else do:
         
      end;
      return;
```

Building the Toolbar

For modularity, we have created a linked routine to build our toolbar.

```lnx
# Fill the toolbar with available data set names...
# Also set up list containing data set names and display...
# action to occur when the data set is displayed in the...
# table editor...
# setools:
#      ddsid = open(trim(dib) || ‘.indexivar’, ‘i’);  /* Index of data sets */
#      call set(ddsid);
#
#-------- Keep track of data sets to switch between......
#-------- and actions to perform on each data set.......
#      dlist = makelist();
#      sunlist = makelist();
#
#-------- Refer to Figure 1 below for the......
#-------- populated list of button information.-------
#      buttonlist = makelist();  /* Make button list */
#      ntoolbar = attr(ddsid, ‘NTOOLBAR’);
#      do i = 1 to ntoolbar;
#         rc = fetchobj(ddsid, i);
#         button = makelist();
#         rc = setitem(button, ‘instrument’, ‘IMAGE’);
#         rc = setitem(button, ‘image’, ‘LABEL’);
#         rc = setitem(button, ‘button’, ‘SPACE’);
#         rc = upcase(trim(member));
#         rc = insr。（sunlist, action, i, dename);
#         end;
#
#-------- Give the toolbar the new list and its size.-----
#-------- Refer to Figure 2 below for the populated-----
#-------- Toolbar referenced by TOOLID.-----
## call send(‘FRAME’, ‘GETSET’, ‘toolid’);
#      rc = setitem(‘toolid’, ‘buttonlist’, ‘SETLIST’);
#      rc = setitem(‘toolid’, ‘ntoolbar’, ‘NUM’);
## call notify(‘toolbar’, ‘UPDATE’);
#      rc = close(ddsid);
#      return;
```

Here is an example of the toolbar button information stored in a list after the iterative loop section shown above ran. Notice that it is a list of lists that will be inserted into the toolbar description information when the SETITEM command is executed. See “display N’ for the contents of the toolbar object after SETITEM is executed. The toolbar attributes that have been inserted for each data set in this particular data subject include the names of the image pictures to be used on the individual toolbar buttons, the label descriptions to appear on the individual toolbar buttons and the number of separator spaces between each button.

```lnx
(31) IMAGE=’SHOW’
   LABEL=’Hires/All’
   SPACE=1
(32) IMAGE=’SHOW’
   LABEL=’Hires/Full’
   SPACE=1
(33) IMAGE=’SHOW’
   LABEL=’Hires/Parttime’
   SPACE=1
(34) IMAGE=’SHOW’
   LABEL=’Open Positions’
   SPACE=1
(35) IMAGE=’ACTIVE’
   LABEL=’Vendor Expenses’
   SPACE=1
(36) IMAGE=’ACTIVE’
   LABEL=’Vendor Expenses’
   SPACE=1
```

FIGURE 1. Attributes of the Data Warehouse Toolbar Buttons
The contents of the toolbar object are displayed below and were captured after the _UPDATE_ method in the previous code section was executed. Information not relevant to this discussion has been removed due to space constraints. The elements of the “buttonlist” list have been inserted and become part of the toolbar’s instance data. Notice also that another instance item “Gray” has now been included as a result of the execution of the _UPDATE_ method. In this example, none of the toolbar buttons will be grayed when the toolbar is built.

```
{ NAME="TOOLBAR" 
  LABEL="TOOLBAR" 
  BTNList: B1< IMAGE="SHOW" 
    LABEL="Hires/All" 
    SPACE=1 
    GRAY="N"
  > 11429
  B2< IMAGE="SHOW" 
    LABEL="Hires/Full" 
    SPACE=1 
    GRAY="N"
  > 11387
  B3< IMAGE="SHOW" 
    LABEL="Hires/Parttime" 
    SPACE=1 
    GRAY="N"
  > 11373
  B4< IMAGE="SHOW" 
    LABEL="Open Positions" 
    SPACE=1 
    GRAY="N"
  > 11369
  B5< IMAGE="ACTIVE" 
    LABEL="Vendor Expenses" 
    SPACE=1 
    GRAY="N"
  > 11355
  } 11403

BTWIDTH=15 
CONFORM=’Y’ _POPULATE_="LISTOF_VALUES” 
LOCATE="" _TYPE_="N” 
NUM=5 
HDR=1 
NDOLS=1 
DESC=’Toolbar’ 
} 8801

FIGURE 2. Attributes of the Toolbar Class for the Data Warehouse.

Constructing the Tabular Display

For the purposes of this paper, the focus of populating the Table Editor will center around displaying the data in a tabular format. This is indicated to the visual display developer by a blank value stored in the ACTION variable in the index of data sets. The DWA can also elect to show the data in a summarized fashion in one of two ways, either by displaying overall totals or by displaying subtotals in addition to the overall totals. Values of the ACTION variable will reflect these requests.

When the DWA determines how to display the data, other decisions are made regarding the presentation of the variables in the data sets. The visual display developer takes all of the information from the index data sets and conditionally executes the SCL code based upon the values of these variables.

In displaying data in a tabular format, it may be desirable to have variables in the columns always visible as users scroll right and left within the table. There may also be variables that will not be displayed in the table. These variable actions are recorded by the DWA in the ACTION variable in the index of data sets.

A value of “HOLD” in the ACTION field of the index data set variables signals the visual display developer which variables should remain visible in the table and the appropriate method call can be sent to the table. Likewise, a value of “HIDE” will indicate which variables will not be displayed in the table, and again an appropriate method call is sent to the table.

In the case of a summarized table, the ACTION variable in the index of data set variables will contain the values, “CLASS” and “ANALYSIS”, communicating to the visual display developer the variables which will be used as CLASS variables and those variables which will be summarized similar to the TABULATE procedure.

Additionally, as the data warehouse data sets are re-created, the DWA positions the variables in the data sets according to the order they are to appear in the table, from left to right. The variable VARNUM in the index of data set variables holds the position number, taken from the CONTENTS procedure output, of the variable in the data set.

“Holding” and “Hiding” Columns in the Table Editor

The subroutine, CHECKCOL, looks for any variables in the index of data set variables that have special actions associated with them and sends the appropriate action methods to the table as to the display or non-display of the variables. This subroutine is called from two places in the SCL program. As shown above, CHECKCOL is executed upon conditional initialization of the table. This code section is also conditionally executed as certain data sets are selected for viewing from the toolbar that appears on the FRAME, if the DWA has made that designation in the index of data sets.

```
CHECKCOL:
    !---------- Information gathered from the INDEX of
              !---------- data set variables(INDEXVAR).
              !---------- Specify column(‘variables) to HOLD when
              !---------- displaying data set in the Table Editor.
    xnum  = sum(index(‘varname’,’.’,’’),1);
    xmember = substr(index(‘varname’),num);
    muncols1 = "upcase(action)”="HOLD”;
    muncols2 = "and member(“’”|’member|)”;
    rc = where(vdoid,muncols1,muncols2);
    rc = clearlist(holdlist);
    rc = clearlist(first);
    rc = clearlist(last);
    !---------- Get position number(VARNUM) of variables
    !---------- in the data set being displayed that are
    !---------- flagged as one to HOLD.
    do while (fetch(vdoid)=-1);
      xvarname = getvart(vdoid,’num’);
      xnum = getvar(vdoid,’num’);
      rc = insert(’holdlist’,’xvarname’,
    !---------- in the table when scrolling right and left.
    if lasten(’holdlist’,’num’)=0 then do:
      rc = sortlist(’holdlist’,’num’,’1’);
    !---------- Get first column to HOLD, others maybe
    if lasten(’first’,’holdlist’,’num’,’1’)=0 then do:
      rc = insert(’first’,’pop’,’holdlist’,’1’);
     More than one
      if lasten(’first’,’pop’,’holdlist’,’1’)=0 then do:
        rc = insert(’last’,’pop’,’holdlist’,’1’);
      if lasten(’first’,’pop’,’holdlist’,’1’)=0 then
call notify(’table’,”SET_HELD_COLUMNS”,”first”);
      else call notify(’table’,”SET_HELD_COLUMNS”,”first”);
    end;
    !---------- Now specify column(‘variables) to HIDE. 
    muncols1 = “upcase(action)”="HIDE”;
    muncols2 = “and member(“’”|’member|)”;
    rc = where(vdoid,muncols1,muncols2);
    do while (fetch(vdoid)=-1);
      xvarname = getvart(vdoid,’num’);
      call notify(’table’,”HIDE_COLUMN”,’xvarname’);
    end;
    return;
```

```
Selecting A Data Set from the Toolbar

The final SCL code extraction consists of the labeled section of code used to select data sets from the toolbar. When a user selects a new data set for viewing, this section executes to send the new data set, along with several other pieces of information including "action requests" to the Table Editor for display.

toolbar:
call notify('toolbar', _GET_LAST(SDL, ndx, selected));

----------- Data set selected from toolbar
/end

if ndx > 0 then do:

/----------- Disconnect data model from table and
    delete its instance before a new instance
    of another data model is created
/end

if xact = ' ' then do:
    .
/end

else if xact = '/' then do:
call notify('table', '"DETACH\_L');
call send(tid, _TERM_);
end:

dename = nameitem(dist, ndx);
xact = getitem(queue, ndx);
call notify('title', _SET\_TEXT_, getitem(dist, ndx));

----------- Prepare for display of next data set
/end

if xact = ' ' then do:
tabid = instance(tabid); / New instance of DATA\_M =
call notify('table', '"ATTACH\_L', tabid);
call send(tid, _SET\_DATA\_GET_, dename);
link checkout;
/end

else do:
    .
/end

return;

Flexibility and usability have been greatly increased by the data driven properties of this data warehouse application. The visual display developer need not be concerned when new data sets are added to the data warehouse. In fact, a rather lengthy period of time passed since this project was finished and work begun on this paper, and the visual display developer was quite surprised to see the number of changes made to the data in several months time! Data sets have been added, others removed, and still others combined as the needs of the users have changed over the past several months. This is really proof that the goals set in the beginning of the project of simplicity and "data-drivenness" were met and continue to be met even now!

THE SAS/WAREHOUSE ADMINISTRATOR PRODUCT

The SAS System with its data access, management, analysis, and presentation capabilities is a powerful tool for SAS users to use in building a data warehouse. The SAS/Warehouser Administrator product, currently being developed, will make it even easier for SAS users to build, maintain, and access a data warehouse by bringing the strengths of the SAS System relative to data warehousing together into one consistent interface plus adding additional capabilities not currently available in other SAS System products.

The features available to users of SAS/Warehouser Administrator are grouped into three broad categories:

Build
All facilities to build and maintain the data warehouse. Building refers to both the initial build of the data warehouse and subsequent iterative modifications and enhancements to the data warehouse. Metadata is created/changed with this feature.

Manage
Data warehouse management capabilities for day-to-day operations. Metadata may be changed with this feature, for example, by the Job Scheduler.

Access
Features that make the data warehouse accessible to end users. Metadata is not changed with this feature.

The DWA of the SAS/Warehouser Administrator product will primarily use the Build and Manage features and occasionally use the Access features. The DSS Analyst and the Business User may use the Build and Manage features, for example, to build a personal data warehouse, but will primarily use the Access features to access existing warehouses built by the DWA.

To facilitate data warehouse definition of data entities and data flow between these entities, some of the Build features are interfaces for:

- defining subjects in the data warehouse
- defining different types of data collections, i.e., detail data, summaries, Data Marts
- identifying sources of operational data
- defining extraction techniques
- defining transformations to transform operational data to warehouse data
- defining different summarization techniques
- populating the target files in the warehouse.

As the warehouse is defined, metadata describing the warehouse entities and relationships will be automatically generated and maintained by SAS/Warehouser Administrator.

The data warehouse environment is organized around the major subjects of a business enterprise, not process-oriented applications. SAS/Warehouser Administrator will permit flexible definition of warehouse data by subject area. Each subject will support a number of different data collections that may reside as SAS data sets or database tables, for example, as an OLAP\_L table. Data is collected into a detail level then summarized into one or more summary groups. These summary groups may then be broken down into time-based summary levels, for example, weekly, monthly, or yearly summaries. Additionally, custom-built data marts may be created from the detail or summary levels of one or more subjects to provide a more departmental or user-specific view of the data. These Data Marts may be stored in any storage format, but will be most useful when stored as a Multi-dimensional Data Base (MDB) for use in data analysis.

SAS/Warehouser Administrator will support access to operational data in the following formats:

- SAS data set or view
- any DBMS supported via a SAS/ACCESS engine
- external files flat files or VSAM.

Operational data sources can be distributed across the network and are accessed using SAS/CONNECT, SAS/SHARE, ODBC, or the 6.11 filename engines (FTP, SOCKET). The DWA can specify whether or not to extract a full copy of the data or just extract the data that has changed since the last extraction. Additionally, other subset criteria may be applied during the extraction phase for data-warehousing purposes. User-written extraction programs will be supported.

After defining operational sources and extraction techniques, a number of different transformation techniques may be chosen to assist the DWA in integrating the warehouse data in a consistent fashion. Tools such as an expression builder, a lookup table builder, and range table checker will permit the DWA to transform data entering the warehouse from an operational source. For example, the DWA will be able to easily register a transformation that converts pounds to kilograms, or converts "M" and "F" to "MALE" and "FEMALE". As with extractions, SAS/Warehouser Administrator users will be able to write their own transformations and register them with the product.

The data summarization component summarizes detail data into day, week, bi-week, month, quarter, and year levels for historical reporting or long-term trend analysis. Each site can use its fiscal time instead of wall calendar time to define levels that are meaningful to
its business. Closely integrated with metadata, the data summarization component allows users to point and click to choose which summary levels to keep and what statistics to maintain for the data. Non-time-based summarization is also supported. Traditional summarization requires that all data that is to be summarized is available at the time of summarization. To remove that limitation, the data summarization component supports incremental summarization that allows warehouse users to easily access many different kinds of summarized results as data is periodically loaded into the warehouse.

Population is the physical loading of data warehouse tables (data, data marts, and archives). SAS/Warehouse Administrator will support either full refresh or append of warehouse tables. The Build features can also be used to build "personal" data warehouses as well as one or more corporate data warehouses. These data warehouses can be centralized on one platform or distributed across multiple platforms.

When the data warehouse is ready for test or production use, the DWA will use the Manage features to facilitate the day to day operations of the data warehouse. Specifically, it provides the DWA with user interfaces for:

- generation of code from metadata definitions
- scheduling of generated code as jobs
- job log viewer to view SAS logs and output
- exception handling to handle data or processing exceptions.

The code generator is called to convert the previously defined metadata into generated SAS code that will load data into the warehouse from the operational sources. That code may then be scheduled and executed as a job via the Job Scheduler.

The Job Scheduler, an independent feature, will be packaged with SAS/Warehouse Administrator. It will enable the scheduling and submission of jobs through a platform independent user interface. Users may use their native host scheduler, such as CRON, or use a SAS-supplied scheduler. A variety of scheduling criteria will be supported, for example, run every 3rd Tuesday, run weekdays only, etc. A job log viewer will be provided to enable the viewing of scheduled or completed jobs scheduled thru the Job Scheduler. Relevant information for each job, for example, return code, will be displayed. The user will be able to view specific job logs and/or output.

Once you build a Data Warehouse, you certainly want business users to access the warehouse easily. So that a business user can find out what data or information a data warehouse contains, the SAS/Warehouse Administrator product will contain a metadata browser that allows the end user to explore the warehouse, looking for subject areas of interest. Once the business users have found subjects of interest, they will most likely want to query those subjects for further information or to perform data analysis. SAS macros will be supplied that will allow the end user to easily access the desired warehouse data from their own programs or applications.

CONCLUSION

Even though more subjects have been added to the data warehouse since its implementation in August of 1995, no major redesign has been required for this data warehouse application at SAS Institute. One of the strongest features of the design of this data warehouse application is that it is portable to any platform that supports the Orlando Release of the SAS System, and at the time of the writing of this paper, plans were being made for the data warehouse to be ported to the PC platform. This version should be ready for display in time for the SUGI21 conference.

Since this data warehouse was developed at SAS Institute, there has been a steady stream of new features and enhancements being added to the SAS System that could further strengthen the power of this data warehouse application. One of the new features is the SAS/Warehouse Administrator product; which will further enhance the data warehouse application at SAS Institute.

The SAS/Warehouse Administrator will give customers a competitive advantage by greatly reducing the time requirements to build and manage a data warehouse. Storing enterprise-wide information in a warehouse framework allows organizations to become much more responsive and customer-driven by turning transaction-oriented data into useful information.

REFERENCES

W.H. Immon(1995), "What is a Data Warehouse?", _Prism Solutions_, 1(1), 1-15


ACKNOWLEDGEMENTS

The authors would like to thank Jim Goodnight, Emily Wallace, Richard Roach, Gail Kramer, and Duane Ressler for their invaluable assistance.

SAS, SAS/AF, SAS/CONNECT, SAS/EIS, SAS/SHARE, and SAS/Warehouse Administrator are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ORACLE is a registered trademark or trademark of Oracle Corporation. © indicates USA registration.

Harriet Janes, SAS Institute Inc., 100 SAS Campus Dr., Cary, NC 27513, 919-677-8000, ext. 7104, e-mail sashmj@vm.sas.com

Shirley Dixon, SAS Institute Inc., 100 SAS Campus Dr., Cary, NC 27513, 919-677-8000, ext. 7926, e-mail sassad@mvs.sas.com

Terry Lewis, SAS Institute Inc., 100 SAS Campus Dr., Cary, NC 27513, 919-677-8000, ext. 7778, e-mail snottll@mvs.sas.com